Task 6

Working With AI

Alexander Garcia

May 2025

Contents

1 Introduction

The goal of this phase is to use tools such as ChatGPT, Claude, etc, to solve the problem and then compare the results with my previous tasks. I used Claude for this since its supposed to be better at coding. I first gave Claude a starting prompt to simply generate a plan with no code. Once the plan was generated in a markdown format, we began checking items off the list in the same way I did, task by task, starting with baseline model then adding things like augmentation and regularization. I told Claude it was the captain and to guide me as I wanted to see its workflow without much more guidance or intervention from myself after the initial plan was laid out.

2 Claude Begins

I started by giving Claude all the information we were presented in the tasks as well as information about the data set, the split I already did, the GPU and env (Google Colab) and the goal of achieving high validation accuracy and testing against test. Claude generated a plan that looked like:

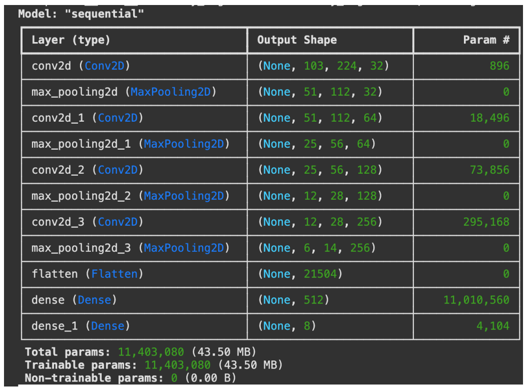
Project Plan

1. **Setup and Data Exploration**
    - Initialize the environment (TensorFlow/Keras)
    - Explore the dataset structure
    - Visualize sample images from each class
    - Analyze class distribution
    - Determine appropriate image dimensions and preprocessing steps

At this point, working with Claude was pretty great. We had a plan, it presented code to be run in blocks since it knew we were working in Colab, and we took it one step at a time. After running 1-3 blocks of code I would post the results back to Claude for analysis. One of the most interesting things it did was run measurements on the images actual height and width and then calculated the ideal dimensions using the median. Its formula for these dimensions are below.

```
suggested_width = int(np.ceil(median_width / 32) * 32)
suggested_height = int(np.ceil(median_height / 32) * 32)
```

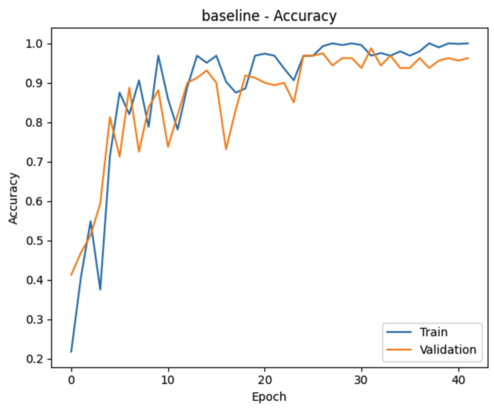
3 Baseline Model



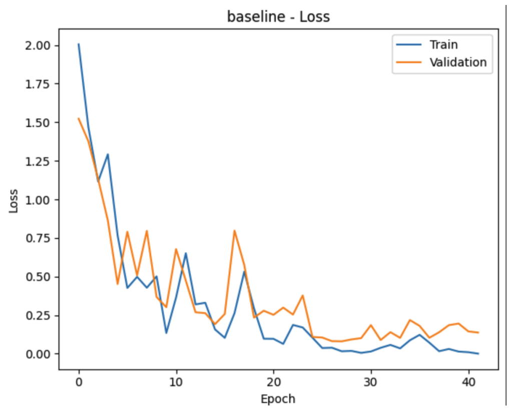
```
Model: "sequential"
┌──────────────────────────────┬─────────────────────────┬────────────┐
│ Layer (type)                 │ Output Shape            │    Param # │
├──────────────────────────────┼─────────────────────────┼────────────┤
│ conv2d (Conv2D)              │ (None, 103, 224, 32)    │        896 │
├──────────────────────────────┼─────────────────────────┼────────────┤
│ max_pooling2d (MaxPooling2D) │ (None, 51, 112, 32)     │          0 │
├──────────────────────────────┼─────────────────────────┼────────────┤
│ conv2d_1 (Conv2D)            │ (None, 51, 112, 64)     │     18,496 │
├──────────────────────────────┼─────────────────────────┼────────────┤
│ max_pooling2d_1 (MaxPooling2D)│ (None, 25, 56, 64)     │          0 │
├──────────────────────────────┼─────────────────────────┼────────────┤
│ conv2d_2 (Conv2D)            │ (None, 25, 56, 128)     │     73,856 │
├──────────────────────────────┼─────────────────────────┼────────────┤
│ max_pooling2d_2 (MaxPooling2D)│ (None, 12, 28, 128)    │          0 │
├──────────────────────────────┼─────────────────────────┼────────────┤
│ conv2d_3 (Conv2D)            │ (None, 12, 28, 256)     │    295,168 │
├──────────────────────────────┼─────────────────────────┼────────────┤
│ max_pooling2d_3 (MaxPooling2D)│ (None, 6, 14, 256)     │          0 │
├──────────────────────────────┼─────────────────────────┼────────────┤
│ flatten (Flatten)            │ (None, 21504)           │          0 │
├──────────────────────────────┼─────────────────────────┼────────────┤
│ dense (Dense)                │ (None, 512)             │ 11,010,560 │
├──────────────────────────────┼─────────────────────────┼────────────┤
│ dense_1 (Dense)              │ (None, 8)               │      4,104 │
└──────────────────────────────┴─────────────────────────┴────────────┘
 Total params: 11,403,080 (43.50 MB)
 Trainable params: 11,403,080 (43.50 MB)
 Non-trainable params: 0 (0.00 B)
```

Figure 1: Claude's baseline model

The baseline model for Claude is a 4 layer CNN with MaxPooling() layers after each Conv that down-sampled by 2. There is a Flatten() layer followed by a hefty Dense layer of 512 neurons that accounts for 96% of the model's total parameters. Early stopping was used with a patience of 10, restore best weights being true, and model checkpoint based on val_accuracy instead of val_loss. I'm curious if the val_accuracy as a monitor was chosen because I told Claude our goal was obtaining high validation accuracy. The baseline model ran for 42 epochs and achieved over 96% accuracy on validation. An interesting thing for me was seeing Claude's model also suffer from training and validation spikes as I tried and failed to get rid of the spikes in accuracy for both training and validation.



(a) Accuracy



(b) Loss

Figure 2: Baseline metrics

3.1 Baseline Metrics

For metrics Claude chose to run classification report and visualize all classes' precision, recall, f1-score, and support.

Class	Precision	Recall	F1-Score	Support
th_10	1.00	1.00	1.00	20
th_11	1.00	1.00	1.00	20
th_12	1.00	0.95	0.97	20
th_13	1.00	1.00	1.00	20
th_14	0.95	1.00	0.98	20
th_15	1.00	1.00	1.00	22
th_16	0.90	1.00	0.95	18
th_17	1.00	0.90	0.95	21

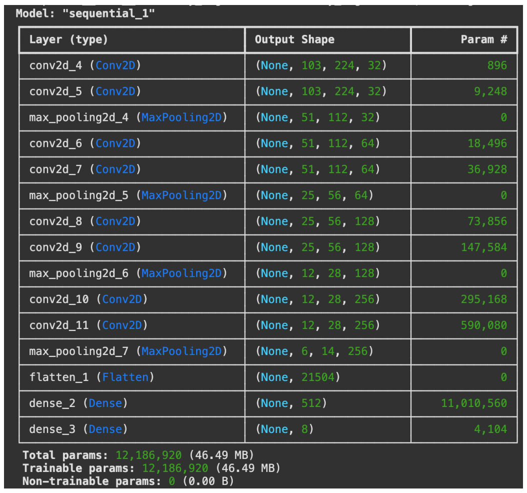
Table 1: Classification Report: Per-Class Metrics

Metric	Precision	Recall	F1-Score	Support
Accuracy	–	–	0.98	161
Macro Avg	0.98	0.98	0.98	161
Weighted Avg	0.98	0.98	0.98	161

Table 2: Classification Report: Summary Metrics

4 Architecture Experiments

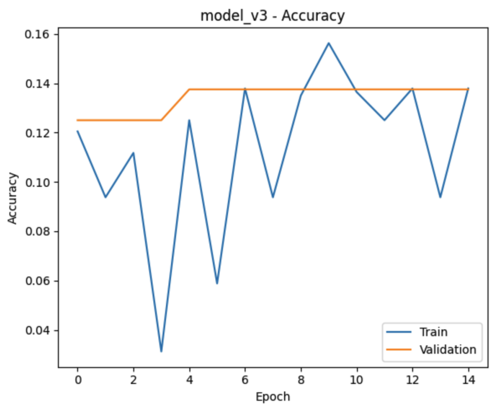
After obtaining a baseline model, Claude went to making a larger network with more convolution layers, stacking 2 convolution layers before MaxPooling. This resulted in a 12.1 million parameter 8 layer model that did not learn very well at all.



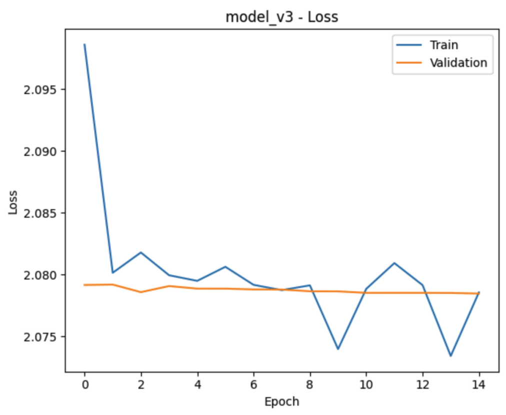
```
Model: "sequential_1"
┌─────────────────────────────────┬────────────────────────┬───────────────┐
│ Layer (type)                    │ Output Shape           │       Param # │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_4 (Conv2D)               │ (None, 103, 224, 32)   │           896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_5 (Conv2D)               │ (None, 103, 224, 32)   │         9,248 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_4 (MaxPooling2D)  │ (None, 51, 112, 32)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_6 (Conv2D)               │ (None, 51, 112, 64)    │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_7 (Conv2D)               │ (None, 51, 112, 64)    │        36,928 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_5 (MaxPooling2D)  │ (None, 25, 56, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_8 (Conv2D)               │ (None, 25, 56, 128)    │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_9 (Conv2D)               │ (None, 25, 56, 128)    │       147,584 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_6 (MaxPooling2D)  │ (None, 12, 28, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_10 (Conv2D)              │ (None, 12, 28, 256)    │       295,168 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_11 (Conv2D)              │ (None, 12, 28, 256)    │       590,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_7 (MaxPooling2D)  │ (None, 6, 14, 256)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_1 (Flatten)             │ (None, 21504)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 512)            │    11,010,560 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 8)              │         4,104 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 12,186,920 (46.49 MB)
 Trainable params: 12,186,920 (46.49 MB)
 Non-trainable params: 0 (0.00 B)
```

Figure 3: Parameters of Claude's second model

At this point Claude decided to stick with the baseline model and attempt augmentation, batch normalization, and then residual connections.



(a) Accuracy



(b) Loss

Figure 4: Claude's alternate 8 layer model

5 Augmentation

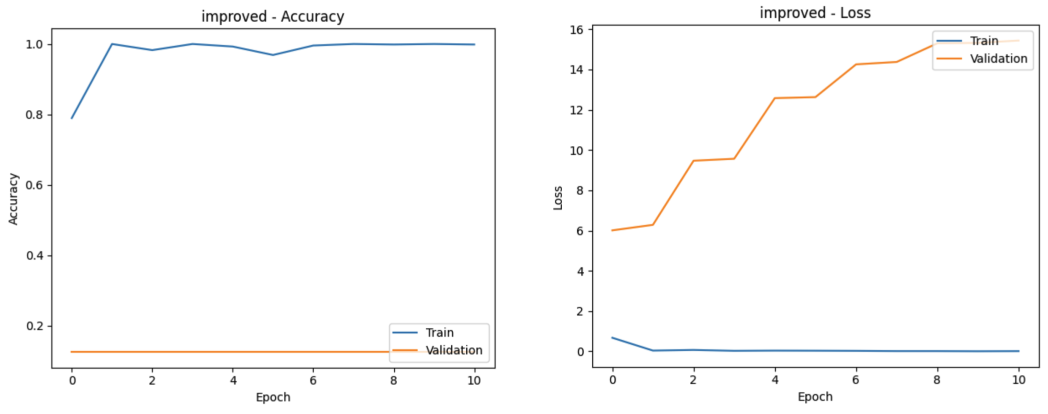
Claude attempted 3 different levels of augmentation starting with base, moderate, and then aggressive. The basic augmentation reached the highest validation accuracy, but dropped by the end of early stopping with the moderate finishing its last epoch with the highest validation accuracy. This was pleasant to see as I also ran into the same effect when trying augmentation

Parameter	**Basic**	**Moderate**	**Aggressive**
rotation_range	10	20	30
width_shift_range	0.1	0.2	0.3
height_shift_range	0.1	0.2	0.3
zoom_range	0.1	0.2	0.3
horizontal_flip	True	True	True
vertical_flip	–	False	True
brightness_range	–	[0.8, 1.2]	[0.7, 1.3]
shear_range	–	–	0.2
fill_mode	–	–	nearest
Validation Accuracy	0.5063	0.6250	0.3500

Table 3: Comparison of Data Augmentation Strategies and Their Validation Accuracy

6 Batch Norm and Regularization

This phase was the most sought after one for myself given the struggles I faced when trying to implement BatchNormalization and regularization. I noticed immediate and intense overfitting which was puzzling for many different models and runs that caused a temporary loss in sanity so it was somewhat reassuring to see Claude suffer the same thing. Claude after trying batch normalization and seeing the results said, “This is fascinating! We have a real mystery on our hands.” Its first attempt was to add BatchNormalization after each convolution which resulted in the metrics below.



(a) Accuracy (b) Loss

Figure 5: Claude’s BatchNormalization attempt

Its at this point Claude starts to break down (so did I, Claude, so did I) because the results of running BatchNormalization, regularization, and Dropout, are resulting in very poor metrics and overfitting. Its final attempt reached 12.5% validation accuracy and with 8 classes that is essentially guessing. Claude even asked to stop running models and instead look at the data. It had me view the validation data to make sure nothing seemed odd and then questioned my use of (224, 103) for image dimensions. Claude forgot at this point that it had created those dimensions which brings me to the major pain point of working with AI - context windows. The chat became too large and switching to a new chat erased Claude’s ”memory”. The plan was no longer in focus and even providing it the markdown plan we created at the beginning didn’t help. It couldn’t remember the baseline model architecture, metrics, preprocessing, everything done before was gone.

7 Conclusion

In summary, I attempted to utilize Claude to build a convolution network putting it through all the tasks in the same order I completed them. The beginning seemed great as it followed a concise plan, printed code in blocks for me to run, then paused for me to give it back results and check items of the list. Everything went south when the context was gone and Claude couldn't get the regularization methods to work. I enjoyed this part the most and found it strangely human because I too suffered at this exact spot with a block in Colab titled "Sanity Check". Claud experienced the same confusion I did and ran through very similar steps; check the data, rerun the best model without any other layers and check its metrics to see if they were real, and question previous decisions. Overall though Claude did beat me with a best validation accuracy of 98.72% and test accuracy of 98.14% while my best against test was 96%.